Technical Feasibility Test for Open Mobile Network Testbed

Topology Gui Environment

Table of Contents

A – Glossary

HTML5	Hyper Text Markup Language Version 5, the standard markup language for creating web pages
RESTful API	Representational State Transfer Application Programming Interface, predominant web API design model
JSON	JavaScript Objection Notation, an open standard format that uses human-readabletext to transmit data objects consisting of attribute–value pair
IDE	Integrated Development Environment, a software application for software development
Eclipse	A famous, open-source IDE
GWT	Google Web Toolkit, an open source set of tools that allows web developers to create and maintain complex JavaScript front-end applications in Java
Chrome	A Google web browser product
MVC	Model-View-Controller, a software architectureused insoftware engineering, that has 3 entities: Model, View and Controller
SVG	Scalable Vector Graphic, XML-basedvector image formatfor two-dimensional graphics that has support for interactivity and animation

I – Introduction

Topology GUI environment (TGE) is a web application that helps illustrate the Open Mobile Network Testbed (OMNT) topology. Users can perform various interactions with the GUI to examine the whole network topology.

To make it portable and working across different platforms, topology GUI environment is developed to give output as HTML5 (the new web standard markup language).

II – System overview

Figure 1 describes the abstract level of how the TGE fits into the whole OMNT. User have a set of user interface (UI) components such as buttons, menus in the TGE to interact with. Base on that, the TGE will use the OMNT cloud Application Programming Interface (API) to get various topology information. The API is implemented as RESTful web API, thus the API will return JavaScript Object Notation (JSON) as its result.

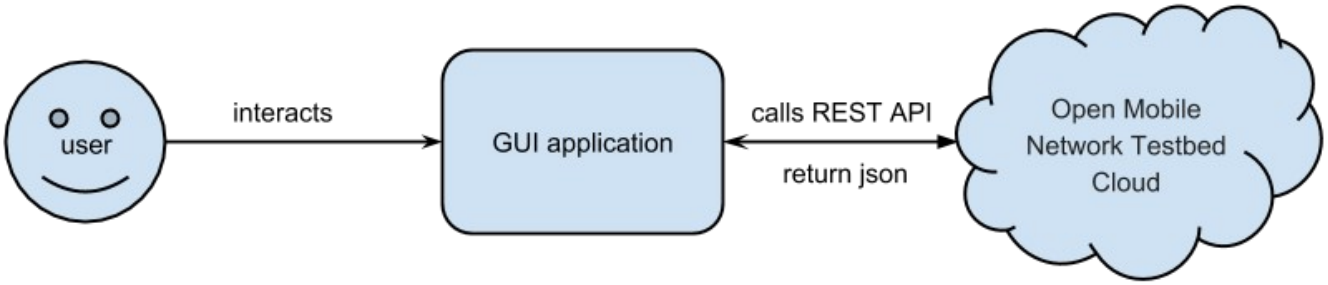

Figure 1: System overview

III – System functionality

The table below describes the set of functionality that user can interact with the TGE

Function	Action
1 – See all OMNT topology information	User clicks on button “get topology”
2 – Zoom into a cluster, get its information	User clicks on “zoom” button, then clicks on a cluster
3 – View a path-flow connection from one end-host/switch (node) to another node in the topology	User clicks on “make path” button, then clicks on two nodes

Table 1: System functionality

IV – Development aspects

1 – GUI design overview

Figure 2 shows the general design of the TGE application. Although it is a web application, the layout is designed to be similar with a normal desktop application including menu bar and task bar, which provides various options to user. For displaying topology information, a left and right panel are used. Both of them can display information in different tabs, where as left panel will display visualized information including image, drawing of shapes, etc, and the right panel will show detail information about the topology such as node IP address, its connections, etc.

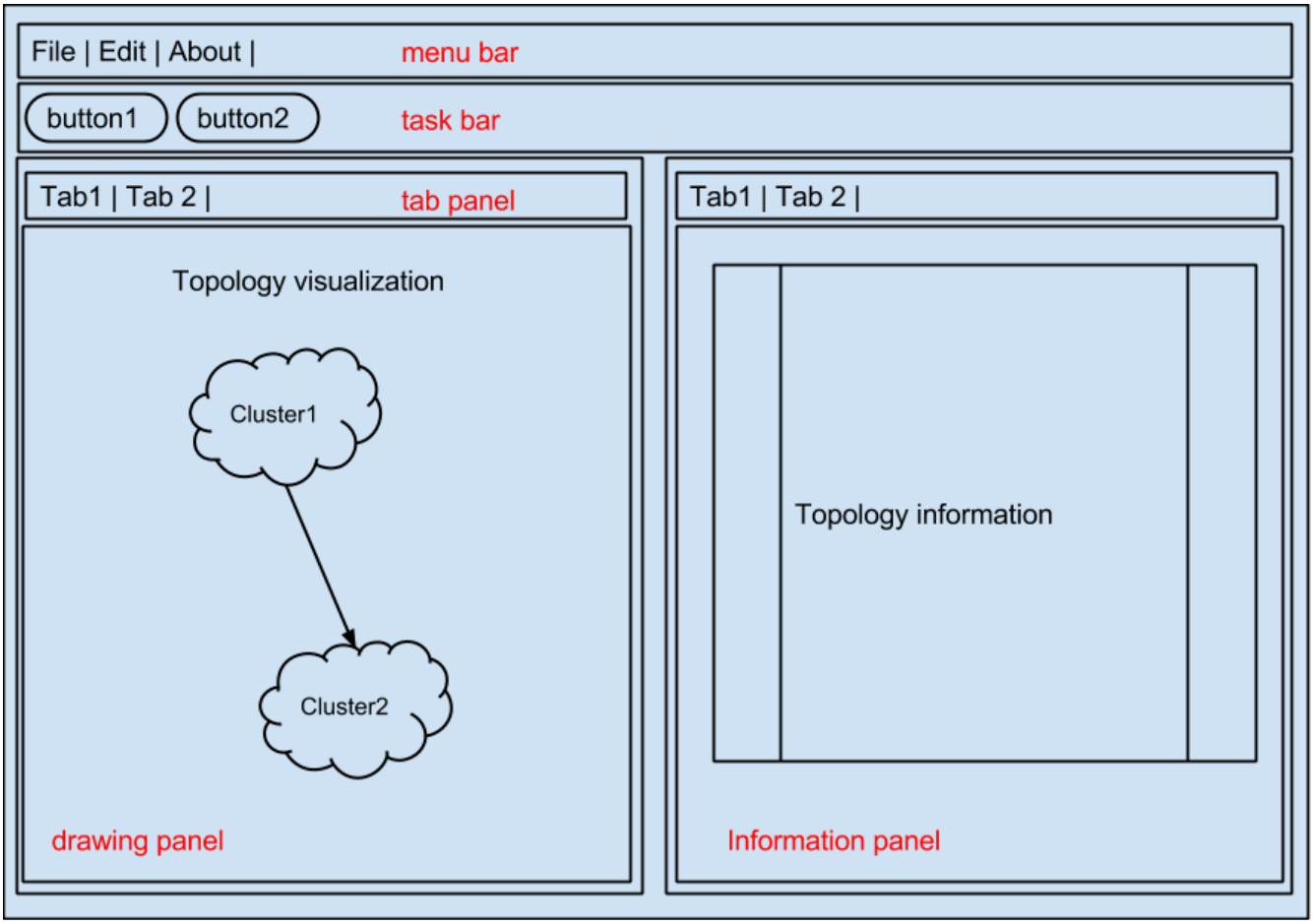

Figure 2: GUI design overview

2 – Development tool/framework

The author uses Google Web Toolkit (GWT) as the framework for development because of the author's proficiency in Java programming language. The chosen Integrated Development Environment (IDE) is Eclipse, an open-source, versatile IDE to help speed up the development process. Other reasons for choosing GWT are the rich set of provided widgets (an UI component) and its strong open-source community. When compiling the source code, GWT will convert Java into JavaScript code to run in the browser. And the main target browser in the development is Google Chrome.

3 – Application implementation

3.1 – MVC Design pattern:

The GUI application has medium complexity in user interaction. It can evolve to a complex GUI application later on, to address this issue, the author applied a modified Model-View-Controller pattern when implementing the application. Figure 3 shows how the modified MVC works in TGE. Each view will have its own controller. When a user interacts with the view, the events will be sent to a separate component called the EventBus. The EventBus will perform some reasoning to deliver the events to appropriate controllers. This model focuses on centralizing the user event processing part to one entity, which can help organize and avoid messy events handling.

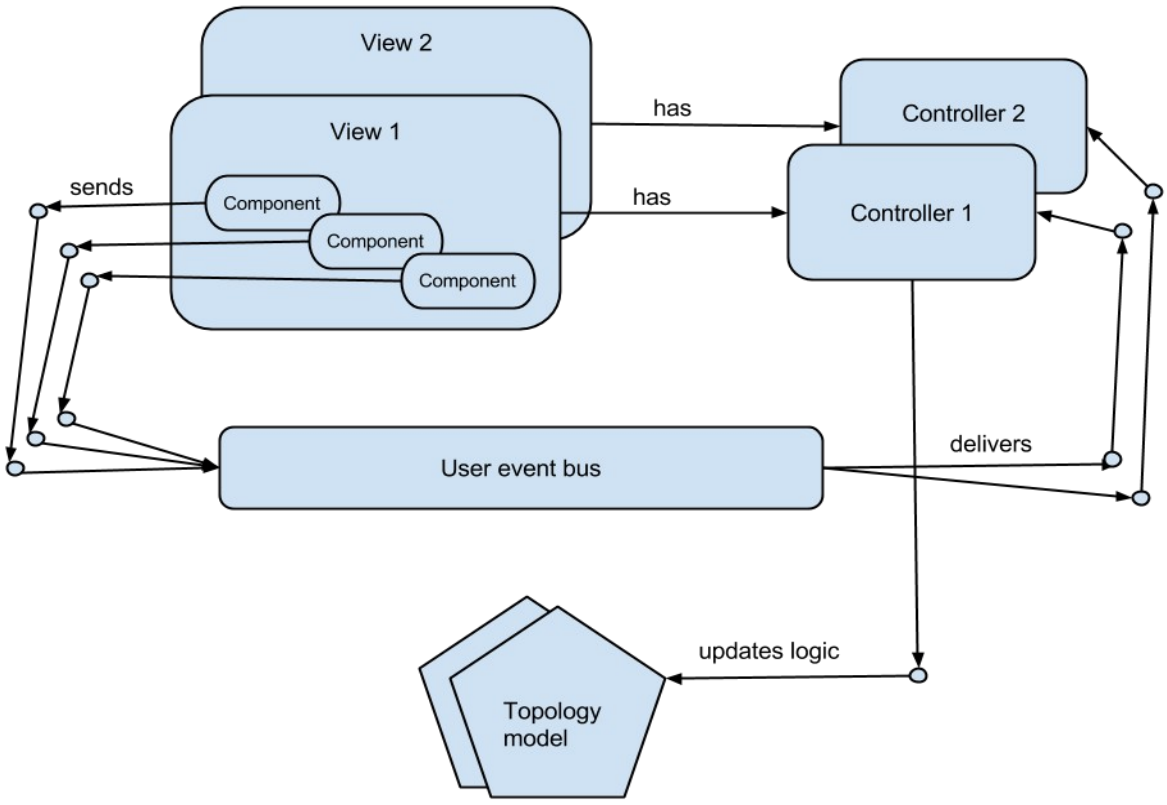

Figure 3: Modified MVC architecture

3.2 – SVG Drawing implementation

For visualized drawing of network topology, the author used Scalable Vector Graphic (SVG), an XML-based vector image format for drawing in the browser. SVG has been supported by many major browsers on the web today. The main benefit of using SVG is that the author can modify the image at runtime to dynamically alter the visualization, which makes the TGE more interactive and attractive.

For example, when a user wants to zoom into a cluster, the author can draw all the nodes inside that cluster as an overlay layer on top of a blurry cluster image in the background, which has been scale up and add blurry attribute at the moment user clicks zoom in. This help user to quickly distinguish all the children nodes are belong to that cluster behind. Figure 4 shows an example of this approach.

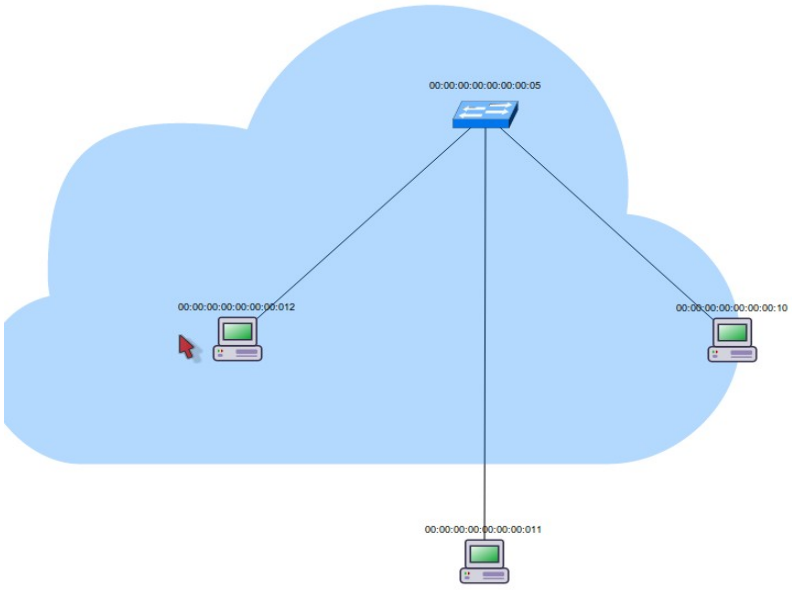

Figure 4: change SVG image at run-time

3.3 – Client/Server communication using RESTful API

When the TGE wants to get network topology information based on user interaction, it will send HTTP requests to the RESTful API service of the OMNT that resided in the cloud. RESTful API help simplify and speed up the the communication between the client and the cloud. JSON is the returned result, which is very lightweight. The TGE will quickly process JSON and give user information. Figure 5 shows an example of JSON result


```
[
    {
        "src-switch": "00:00:00:00:00:00:00:02",
        "src-port": 3,
        "dst-switch": "00:00:00:00:00:00:00:03",
        "dst-port": 2,
        "type": "internal",
        "direction": "bidirectional"
    },
    {
        "src-switch": "00:00:00:00:00:00:00:02",
        "src-port": 4,
        "dst-switch": "00:00:00:00:00:00:00:04",
        "dst-port": 2,
        "type": "internal",
        "direction": "bidirectional"
    },
    {
        "src-switch": "00:00:00:00:00:00:00:04",
        "src-port": 1,
        "dst-switch": "00:00:00:00:00:00:00:06",
        "dst-port": 1,
        "type": "internal",
        "direction": "bidirectional"
    }
]
```


Figure 5: Example JSON result